ИНФОРМАТИКА, ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА И УПРАВЛЕНИЕ
INFORMATION TECHNOLOGY, COMPUTER SCIENCE, AND MANAGEMENT





УДК 004.42 https://doi.org/10.23947/2687-1653-2021-21-2-200-206



Теоретические основы организации ветвлений и повторений в программах на языке логического программирования Пролог

Д. В. Здор
ФГБОУ ВО «Приморская государственная сельскохозяйственная академия» (г. Уссурийск, Российская Федерация)

Введение. Рассматривается организация ветвлений и повторений в контексте логического программирования на примере языка Пролог. Принципиальной особенностью программы на языке логического программирования является то, что компьютер должен решать задачу, проводя рассуждения подобно человеку. Такая программа содержит описание объектов и отношений между ними на языке математической логики. При этом остается актуальной программная реализация ветвлений и повторений в условиях отсутствия в логическом языке специальных операторов для указанных конструкций. Целями работы явились выявление наиболее эффективных способов для решения задач с применением ветвлений и повторений средствами языка логического программирования Пролог, а также демонстрация полученных результатов на примерах вычислительных задач.

Материалы и методы. Выполнен анализ специальной литературы по теме исследования. Использованы методы обобщения, систематизации знаний, тестирования программы, анализ хода ее выполнения.

Результаты исследования. Предложены конструкции организации ветвлений и повторений в программе на языке Пролог. Для организации повторений предложены различные варианты завершения рекурсивного цикла при решении задач.

Обсуждение и заключения. Рассмотрены способы организации ветвлений и повторений на языке логического программирования Пролог. Все эти способы продемонстрированы на примерах решения задач вычислительного характера. Полученные результаты могут быть использованы при дальнейшей разработке рекурсивных предикатов в логических языках программирования, а также в учебном процессе при изучении логического программирования на языке Пролог. Приведенные примеры программ позволяют использовать их в качестве технологической основы программирования ветвлений и повторений на языке логического программирования Пролог.

Ключевые слова: логическое программирование, ветвление, повторение предикатов, рекурсивное правило, условие окончания рекурсии.

Для цитирования: Здор, Д. В. Теоретические основы организации ветвлений и повторений в программах на языке логического программирования Пролог / Д. В. Здор // Advanced Engineering Research. — 2021. — Т. 21, № 2. — С.200–206. https://doi.org/10.23947/2687-1653-2021-21-2-200-206



Theoretical foundations of the organization of branches and repetitions in programs in the logic programming language Prolog

D. V. Zdor
Primorskaya State Academy of Agriculture (Ussuriysk, Russian Federation)

Introduction. The organization of branches and repetitions in the context of logical programming is considered by an example of the Prolog language. The fundamental feature of the program in a logical programming language is the fact that a computer must solve a problem by reasoning like a human. Such a program contains a description of objects and relations between them in the language of mathematical logic. At the same time, the software implementation of

branching and repetition remains a challenge in the absence of special operators for the indicated constructions in the logical language. The objectives of the study are to identify the most effective ways to solve problems using branching and repetition by means of the logic programming language Prolog, as well as to demonstrate the results obtained by examples of computational problems.

Materials and Methods. An analysis of the literature on the subject of the study was carried out. Methods of generalization and systematization of knowledge, of the program testing, and analysis of the program execution were used.

Results. Constructions of branching and repetition organization in a Prolog program are proposed. To organize repetitions, various options for completing a recursive cycle when solving problems are given.

Discussion and Conclusions. The methods of organizing branches and repetitions in the logic programming language Prolog are considered. All these methods are illustrated by examples of solving computational problems. The results obtained can be used in the further development of the recursive predicates in logical programming languages, as well as in the educational process when studying logical programming in the Prolog language. The examples of programs given in the paper provide using them as a technological basis for programming branches and repetitions in the logic programming language Prolog.

Keywords: logical programming, branching, repetition of predicates, recursive rule, recursion termination condition.

For citation: D. V. Zdor. Theoretical foundations of the organization of branches and repetitions in programs in the logic programming language Prolog. Advanced Engineering Research, 2021, vol. 21, no. 2, pp. 200–206. https://doi.org/10.23947/2687-1653-2021-21-2-200-206

Введение. Языки логического программирования применяются в качестве инструментария решения задач в области построения систем искусственного интеллекта [1]. Одним из языков непроцедурного логического программирования является Пролог. Программа на этом языке использует теорию исчисления предикатов, представляет собой последовательность фактов и правил, с помощью которых описываются объекты и задаются отношения между ними. Затем формулируется цель, являющаяся утверждением, которое должно быть доказано в ходе выполнения программы. Механизм выполнения программы основывается на осуществлении попытки доказательства цели на основе фактов и правил программы с помощью стандартного механизма сопоставления и поиска с возвратом [2].

Указанные обстоятельства в значительной степени влияют на подход, применяемый при составлении программы. В традиционном процедурном программировании в основе построения программы лежит алгоритм решения задачи. Процедурные языки позволяют программно реализовать составленный алгоритм с помощью операторов. При этом структурные процедурные языки программирования позволяют реализовать базовые алгоритмические структуры. Объектно-ориентированное программирование является эволюционным продолжением развития такой технологии. В основе программы лежит объект, его свойства, методы и события. Однако следует заметить, что обработчик события, представляющий собой процедуру, выполняемую при наступлении события в объектно-ориентированных языках программирования, также содержит последовательность операторов, решающих определенную задачу. Здесь же, на языке логического программирования Пролог, необходимо описать задачу и задать правила ее решения на языке математической логики с учетом особенностей механизма выполнения программы [3].

Таким образом, составление программы для решения простой вычислительной задачи на языке Пролог потребует освоения особого подхода, связанного со спецификой логического программирования. Несмотря на то, что язык Пролог нашел основное применение в области построения экспертных систем, программная реализация вычислений также является актуальной задачей, поскольку вычислительные задачи входят в качестве элементов систем обработки информации, в том числе интеллектуальных [4].

Различным аспектам программирования на языке Пролог посвящен ряд работ. В работе Адама Лалли и Пола Фодора описываются правила сопоставления с образцами на языке программирования Пролог. В частности, авторы предлагают использовать обратное отслеживание вместо сопоставления с образцом, так как необходимо проверять множество условий при синтаксическом анализе, то есть возникает потребность в таком языке запросов, в котором можно включать или исключать условия в зависимости от некоторого контекста. Язык Пролог признан эффективным решением проблемы сопоставления с образцом, а также проблем поиска в глубину и отслеживания с возвратом. Исследователи считают, что несмотря на свою простоту, язык Пролог очень выразителен и позволяет рекурсивным правилам представлять достижимость в деревьях синтаксического анализа, выполнять и воспринимать операцию отрицания в качестве сбоя для проверки отсутствия условий [5].

Н. И. Цуканова детально рассматривает предметную область использования программирования посредством логических моделей, демонстрирует связь базовых логических понятий предикатов с базовыми

языковыми конструкциями Пролога. В работе подробно рассматриваются основы логического программирования и структура программы, основные алгоритмы, при этом в качестве примера используется язык Visual Prolog 7 [6].

Э. Коста рассматривает Пролог как язык логического программирования, также подробно описывает его основные конструкции, структуру программы, виды предложений на Прологе, особенности выполнения программы [7].

И. Братко рассматривает алгоритмы искусственного интеллекта на языке Пролог. Главная ценность работы заключается в том, что исследователь демонстрирует использование языка Пролог в различных областях искусственного интеллекта, в том числе для выполнения эвристического поиска, программирования в условиях ограничений, машинного обучения и т. д. [8].

Решение логических задач на языке программирования Пролог изучено в работе А. Н. Адаменко. Она имеет принципиальное значение, так как автор рассматривает рекурсию как способ организации повтора предикатов [9].

В работе Тарушкина В. Т., Тарушкина П. В., Тарушкиной Л. Т. и Юркова А. В. рассматривается логика предикатов и языка Пролог, что позволяет сделать ряд ценных теоретических выводов об основах использования программного языка [10].

Солдатова О. П. и Лёзина И. В. рассматривают язык Пролог как элемент логического программирования и аксиоматических систем. Также в работе рассматриваются основные стратегии решения задач, процедурность программы, в частности повторения и рекурсия языка [11].

Майкл А. Ковингтон, Роберто Баньяра, Ричард А. О'киф, Ян Вилемекер и Саймон Прайс рассматривают проблему кодирования на языке Пролог. Работа исследователей содержит рекомендации по разметке кода, соглашениям об именах, документации, правильному использованию функций Пролога, разработке программ, отладке и тестированию. В каждом руководстве представлено его обоснование, а там, где есть спорные моменты, представлены иллюстрации относительных плюсов и минусов каждой альтернативы [12].

Марков В. Н. представляет обзор новых и традиционных инструментов логического программирования, а также разбирает основные парадигмы функционального программирования, которое органически реализовано в версии Visual Prolog 7.5. Также рассмотрены основные способы обработки и дальнейшего представления массивов, ветвлений, повторов, графов [13].

Гупта Г., Понтелли Е., Али К. А. М., Карлссон М., Эрменегильдо М. В. в своей работе представляют всесторонний обзор проблем параллельного выполнении языков логического программирования, а также наиболее актуальных подходов. Основное внимание исследователи уделяют проблемам, возникающим при параллельном выполнении программ Prolog, в частности организации ветвлений и повторений. В статье описываются основные методы параллелизма и параллелизма с общей памятью, а также их комбинаций [14].

Также необходимо отметить работу Половиковой О. Н., Ширяева В. В., Оскорбина Н. М., Смоляковой Л. Л., где анализируются особенности выполнения логических задач на языке Пролог, в частности подход в поиске ответов на базе генерации состояния и процедуры проверки, в ходе которых совершаются повторы и организуются ответвления. В работе представлено решение логической задачи, которое на практике иллюстрирует разработанный исследователями подход [15].

Анализ показал, что в литературных источниках рассмотрено решение задач вычислительного характера, в которых необходимо согласно алгоритму реализовать ветвления либо повторения. Во всех проанализированных работах имеются прямые или косвенные описания организации ветвлений и повторений в Прологе. Однако отсутствуют данные о максимальной рациональности какого-либо подхода, что свидетельствует о неполноте знаний по применению языка и его возможностях при составлении программ в контексте логического программирования.

Цель данного исследования заключается в определении наиболее рациональной теоретической основы организации ветвлений и повторений в программах на языке логического программирования Пролог.

Материалы и методы. Выполнен анализ специализированной литературы по теме исследования за последние 15 лет. Использовались методы сравнительного анализа, обобщения и систематизации знаний, тестирования программы, анализа хода выполнения программы.

Результаты исследования. На основе анализа литературных источников определён наиболее рациональный метод организации ветвлений и повторений в программах на языке логического программирования Пролог. Сформулирована теоретическая основа организации ветвлений и повторений в программах на этом языке. Необходимо отметить, что процесс доказательства цели программы сводится к сопоставлению утверждений, входящих в цель, с фактами и правилами из базы знаний программы. Истинность

утверждений, входящих в цель, устанавливается в порядке их следования в цели слева направо. При этом сопоставление с предикатами базы знаний программы осуществляется в порядке их следования сверху вниз. Если на каком-то шаге выполнения программы, т. е. доказательства решения задачи, сопоставление какого-то предиката терпит неуспех, система Пролог использует второй процесс — поиск с возвратом.

Для управления поиском решения в Прологе имеется ряд стандартных предикатов, позволяющих изменять стандартный механизм поиска. К таким предикатам относится предикат fail, имеющий значение «ложь», и предикат отсечения возвратов, его можно записать как знак «!».

При организации ветвления необходимо составить его конструкцию. В качестве хвостовых предикатов в правилах, составляющих конструкцию ветвления, необходимо включить взаимоисключающие условия. Рассмотрим пример нахождения наибольшего из двух чисел. Известно, что максимум из 2-х чисел в математике можно определить следующим образом:

$$\max(x, y) = \begin{cases} x, & x \geq y \\ y, & x < y \end{cases}.$$

На Прологе нахождение max (x, y) можно задать так:

predicates
max (***real, real, real)*** /* 3-хместныйпредикат */
clauses,
max (X, Y, X) :- X > =Y.
max (X, Y, Y) :- X<Y.

База знаний не содержит фактов, а содержит два правила. Первое правило можно прочитать так: «Максимальным из значений X, У является X, если X > = Y», второе правило: «Максимальным из значений X, У является У, если X < Y».

Если задать внешнюю цель:
goal
max (15, 10, Max).   /* X =15, У =10 */
пролог найдет решение:
Max = 15
1 solutions.

При нахождении решения Прологу потребовалось только первое правило, т. к. при заданных значениях X > У. Так как цель внешняя, Пролог проверит и второе правило на предмет нахождения другого решения и потратит на это время. Условия в правилах взаимоисключающие, поэтому, если добавить предикат отсечения:

max (X, Y, X) :- X > =Y, !.   или   max (X, Y, Y) :- X <Y, !.
max (X, Y, Y) :- X <Y.   max (X, Y, X) :- X > =Y.

Пролог, найдя максимальное значение по первому правилу, уже не будет проверять второе. В данном примере наличие предиката отсечения обусловлено логикой программы.

Рассмотрим способы организации повторений в программе на языке Пролог. Для организации повтора действий необходимо задать циклическую конструкцию:

repeat.
repeat:- repeat.

Такая конструкция задает бесконечный цикл. В качестве предиката цикла можно использовать предикат с любым именем. Предикат ***repeat*** в данном случае не является стандартным и должен быть описан в разделе ***predicates***. В программе необходимо предусмотреть выход из циклической конструкции.

Рассмотрим в качестве примера программу нахождения значений функции $y = x^2$ для аргументов, которые пользователь вводит последовательно с клавиатуры:

predicates
tab
repeat
test(real)
clauses
repeat.
repeat:-repeat.
tab:-repeat, readreal(X), test(X).
test(10e10):-nl.
test(X):-Y=X*X, write("X=",X," Y=",Y), nl, fail.
goaltab.

Если с клавиатуры вводить числа, то на экране будет осуществляться вывод значений аргумента, а рядом — вычисленное значение функции. Выполнение программы завершится, если будет введено число в экспоненциальной форме 10e10. До тех пор, пока не будет введено указанное число, хвостовой предикат test(X) будет иметь значение «ложь» из-за наличия предиката ***fail*** в хвосте правила test(X). В этом случае Пролог использует механизм поиска с возвратом, а предикат ***repeat*** зацикливает процесс на ввод нового числа. Если ввести число в экспоненциальной форме 10e10, цель будет доказана.

В данном примере организация повторений реализована через конструкцию с использованием правила ***repeat:- repeat***, которая, по сути, является рекурсивной, так как и в голове, и в теле имеется предикат с одинаковым именем.

Теперь рассмотрим организацию повторений непосредственно с помощью рекурсивного правила на примере задачи табулирования функции $y = x^2$ на отрезке $\left[a;b\right]$ с шагом h:

```
predicates
tab(real,real,real)
clauses
tab(A,B,H):-A<=B, X=A, Y=X*X, write("X=",X," Y=",Y),nl,
        A1=A+H, B1=B, H1=H, tab(A1,B1,H1).
goal
write("a="), readreal(A), write("b="), readreal(B),
write("h="), readreal(H), nl, tab(A,B,H).
```

При исполнении программы на экран будет выведена таблица значений функции на заданном пользователем отрезке с заданным шагом.

Данная программа содержит рекурсивное правило tab(A,B,H). Для выхода из рекурсии в программе использовано хвостовое условие A<=B. Недостатком данного способа выхода из рекурсии является то обстоятельство, что предикат цели tab(A,B,H) после выполнения программы будет иметь значение «ложь» из-за хвостового условия A<=B в теле рекурсивного правила tab(A,B,H) после достижения переменной A значения, превосходящего значение переменной B.

Этот недостаток можно исправить, добавив в программу нерекурсивное правило с тем же именем, что и рекурсивное. Программа примет следующий вид:

```
predicates
tab(real,real,real)
clauses
tab(A,B,_):-A>B.
tab(A,B,H):-A<=B, X=A, Y=X*X, write("X=",X," Y=",Y),nl,
        A1=A+H, B1=B, H1=H, tab(A1,B1,H1).
goal
write("a="), readreal(A), write("b="), readreal(B),
write("h="), readreal(H), nl, tab(A,B,H).
```

Отметим, что в этом случае в рекурсивном правиле, отвечающем за повторения предикатов, и в нерекурсивном, отвечающем за корректный выход из рекурсии, используются взаимоисключающие условия.

Приведем еще один пример организации повторения с помощью рекурсивного правила. Рассмотрим программу вычисления суммы k первых членов ряда $S = 1 + x + x^2 + x^3 + \ldots + x^n + \ldots$:

```
domains
k=integer
x=real
n=real
s=real
predicates
sum(x,k,n,s)
clauses
sum(_,1,N,S):-N=1, S=N.
sum(X,K,N,S):-K1=K-1, sum(X,K1,N1,S1), N=N1*X, S=S1+N.
goal
write("X="), readreal(X),
write("K="), readint(K),
sum(X,K,N,S), nl,
```

write("S=",S).

В этой программе k — количество членов ряда, сумму которых необходимо вычислить. Переменная n необходима для нахождения i -го члена ряда, в этой переменной находится искомая сумма.

При попытке доказательства утверждения цели sum(X,K,N,S) будет выполнятся рекурсивное правило sum(X,K,N,S), при условии, что $K \neq 1$, в качестве хвостового условия которого содержится рекурсивный вызов sum(X,K1,N1,S1), при этом значение переменной $K1 = K - 1$. Так, значение переменной k будет уменьшаться до 1 при прямом ходе рекурсии. После сопоставления с правилом sum(_,1,N,S):-N=1, S=N начнется обратный ход рекурсии, на каждом витке которого будет вычисляться следующий член ряда путем умножения предыдущего члена на значение аргумента x. Значение суммы ряда будет накапливаться в переменной s.

Итак, организация повторений в программе на Прологе реализуется с помощью рекурсии. Рекурсивное правило задает бесконечный цикл. Этот цикл применяется в целях решения вычислительной задачи. При этом завершение цикла и выход из рекурсии может быть реализован разными способами:

— ввод пользователем с клавиатуры заранее предусмотренного значения переменной;

— использование логического выражения в качестве одного из условий рекурсивного правила;

— использование нерекурсивного правила с тем же именем, что и рекурсивное, но в качестве предикатов тела в этом правиле должны выступать истинные утверждения.

Обсуждение и заключения. Представлены наиболее эффективные способы организации ветвлений и повторений на языке логического программирования Пролог, выявленные в результате сравнительного анализа литературных источников. Приведенные задачи имеют учебно-тренировочное назначение. Их решение способствует пониманию теории логического программирования на основе применения аналогии с учетом специфики построения программы на языке Пролог и особого механизма ее выполнения.

Приведенные примеры программ позволяют использовать их в качестве технологической основы программирования ветвлений и повторений на языке логического программирования Пролог. Полученные результаты могут быть использованы в дальнейшей разработке использования рекурсивных предикатов в логических языках программирования, а также в учебном процессе при изучении логического программирования на языке Пролог.

Библиографический список

1. Genesys-Pro: Innovations in Test Program Generation for Functional Processor Verification / E. Adir, L. Almog, E. Fournier [et al.] // IEEE Design & Test of Computers. — 2014. — Vol. 21 (2). — P. 84–93. https://doi.org/10.1109/MDT.2004.1277900

2. ARM Architecture Reference Manual. ARM DDI 0487A.f. ARM Corporation, 2015. — 5886 p.

3. Kent D. Lee. Foundations of Programming Languages / Kent D. Lee // Springer, 2017. — 370 p.

4. Ute Schmid. Inductive Synthesis of Functional Programs: Universal Planning, Folding of Finite Programs, and Schema Abstraction by Analogical Reasoning / Ute Schmid // Springer Science & Business Media, 2013.

5. Adam Lally. Natural Language Processing with Prolog in the IBM Watson System / Adam Lally, Paul Fodor // Association for Logic Programming, 2011.

6. Цуканова, Н. И. Теория и практика логического программирования на языке Visual Prolog 7 / Н. И. Цуканова, Т. А. Дмитриева. — Москва : Горячая линия – Телеком, 2013. — 232 с.

7. Eduardo Costa. Visual Prolog 7.3 for Tyros / Eduardo Costa // 2010. — 270 p. — URL: http://visual-prolog.com/download/73/books/tyros/tyros73.pdf

8. Братко, И. Алгоритмы искусственного интеллекта на языке Prolog / И. Братко ; [пер. с англ. К. А. Птицина]. — 3-е изд. — Москва : Вильямс, 2004. — 637 с.

9. Адаменко, А. Н. Логическое программирование и Visual Prolog / А. Н. Адаменко, А. М. Кучуков. — Санкт-Петербург: БХВ-Петербург, 2003. — 982 с.

10. Логика предикатов и язык Пролог / В. Т. Тарушкин, П. В. Тарушкин, Л. Т. Тарушкина, А. В. Юрков // Современные наукоемкие технологии. — 2010. — № 4. — С. 62–63.

11. Солдатова, О. П. Программирование на языке ПРОЛОГ / О. П. Солдатова, И. В. Лёзина. — Самара : Репозитарий Самарского государственного аэрокосмического университета : [сайт]. — 2008. — 52 с.

12. Coding guidelines for Prolog / Michael A. Covington, Roberto Bagnara, Richard A. O'Keefe [et al.] // Theory and Practice of Logic Programming. — 2011. — Vol. 12 (6). — P. 889–927.

13. Марков, В. Н. Современное логическое программирование на языке Visual Prolog 7/5 / В. Н. Марков. — Санкт-Петербург : БХВ-Петербург, 2016. — 541 с.

14. Parallel Execution of Prolog Programs: a Survey / G. Gupta, E. Pontelli, K.A.M. Ali [et al.] // ACM Transactions on Programming Languages and Systems. — 2011. — Vol. 23 (4). — P. 472.

15. Особенности программной реализации логических задач на языке PROLOG / О. Н. Половикова, В. В. Ширяев, Н. М. Оскорбин, Л. Л. Смолякова // Известия Алтайского государственного университета. — 2021. — № 1 (117). — С. 166–120.

Поступила в редакцию 01.04.2021
Поступила после рецензирования 14.04.2021
Принята к публикации 04.06.2021

Об авторе:

Здор Дмитрий Валерьевич, доцент инженерно-технологического института, ФГБОУ ВО «Приморская государственная сельскохозяйственная академия» (692500, РФ, Приморский край, г. Уссурийск, пр. Блюхера, 4), кандидат педагогических наук, доцент, ORCID: https://orcid.org/0000-0003-1131-6708, jevgeniya.999.gn@mail.ru

Автор прочитал и одобрил окончательный вариант рукописи.